# CA4003 – Compiler Construction Assignment 1

**Declaration on Plagiarism**

**Assignment Submission Form**

  This form must be filled in and completed by the student(s) submitting an assignment

|  |
| --- |
| Name(s):  **Michael O’Hara** |
| Programme: **Compiler Construction Assignment** |
| Module Code: **CA4003** |
| Assignment Title: **CCALL Language Parser** |
| Submission Date: **3/11/2019** |
| Module Coordinator:  **David Sinclair** |

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all predicates, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I/We have read and understood the referencing guidelines found at

<http://www.dcu.ie/info/regulations/plagiarism.shtml>, <https://www4.dcu.ie/students/az/plagiarism>and/or recommended in the assignment guidelines.

Name(s): \_\_\_\_Michael O’Hara\_\_\_\_\_\_\_\_\_ Date: \_\_03/11/19\_\_\_\_\_\_

The purpose of this report is going through and breaking down the process used in creating the lexical and syntax analyser for the CCAL language. The first thing I did was refer to the Lexical Analyser notes from class. After reading over these I took the template provided in the notes and laid out my JAVACC file to have 4 sections. Which are

1. Options
2. User Code
3. Token Definitions
4. Grammar & Production Rules

I also made several input files which contain the definitions given at the end of the CCAL description PDF. These allowed to test if my CCALTokeniser was working.

Options:

The options for this language are quite short as we are told the language is not case sensitive.

*options {*

*IGNORE\_CASE = true;*

*}*

User Code:

The next section was the user code which must begin by declaring the name of the parser, I called my parser CCALTokeniser.

*PARSER\_BEGIN(CCALTokeniser)*

Then the class needs to be called the same name as the parser so in my case it will be:

*public class CCALTokeniser*

The rest of the user code initialises the parser if the input file is passed as command line arg. If the input file is not correct or another error occurs, it will return the correct syntax to run the file. There are also a few lines at the end which return if the file parsed correctly or did not parse correctly

Token Definitions:

The next section consists of the definitions for the language CCAL. All these definitions came from the CCAL.pdf file provided. It was just a matter of going in and adding in each definition. Each type of token is denoted by a comment Title. This was to make it easier to find and edit the tokens if the need ever arose.

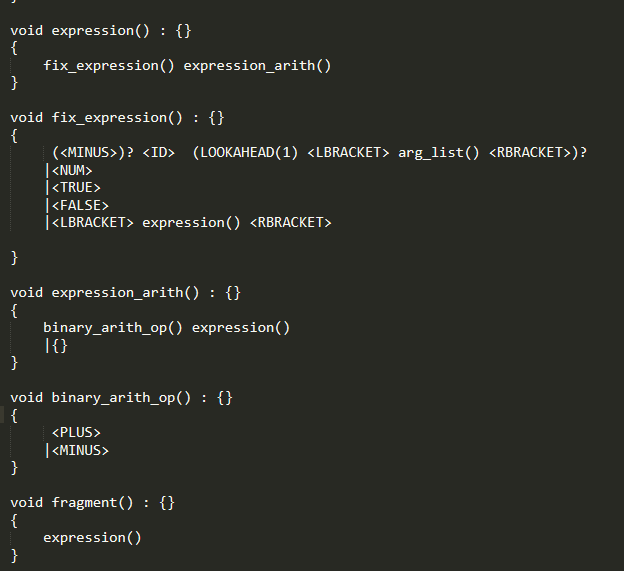
This need did arise when defining the <SKIP> token. It clashed with the regular expression that the CCAL language used for skipping characters. The method I used to get around this was to call it <SKIPTOKEN>

Grammar & Production Rules:

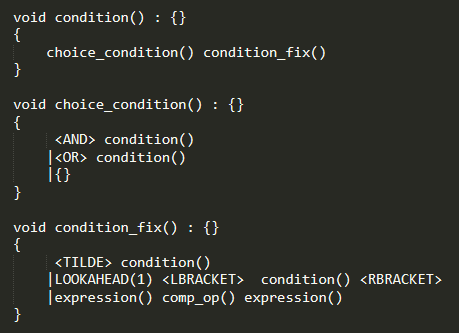
The production rules for this parser were written to correspond to the ones given in the CCAL.pdf. They were later edited to resolve any errors that were found during the compilation and later running of the parser. Any occurrences of the symbol that denotes epsilon (**ϵ**) has been replaced with {} which denotes the empty set.

After the production rules were written I went back to fix any syntax errors that were introduced while writing the grammar. I then had to deal with the 2 left recursion warnings that found in the condition function and in the expression function.

The left recursion in the expression function ( expression → function → expression) I put the definition in the fragment function into the expression function the and then called expression inside of fragment and this allowed me to clearly identify the alpha and beta in the function and then apply the method to eliminate the left recursion



The left recursion in the condition function (condition → condition) was fixed by splitting up the condition function and then editing the original function



Choice Conflicts:

Due to the ambiguous nature of the CCAL language there was a number of choice conflicts that would need to be resolved in order to be able to parse files. This was done by using lookaheads. There was a few ways of doing this such as using a higher number of look ahead such as LOOKAHEAD(2) or even LOOKAHEAD(3) but I found this to be unnecessary as if the LOOKAHEAD is placed in an effective spot then it only needs to be a lookahead of 1.

Running the files:

The first thing that needs to be done is to compile the “Assignment1.javacc” by running the command “*javacc Assignment1.javacc”.* Doing so will then create any of the necessary files needed. The next this step is to compile the file called “CCALTokeniser.java2 by running the command *“javac CCALTokeniser.java”.* The final step is to run the run the parser by running the command “*java CCALTokeniser”.* You can also read in a file for the parser to pass by running *“java CCALTokeniser (someinputfile)”*